Criando Gradle Plugins para automatizar tarefas em projetos multi-módulos

...

TDC Connections - Trilha Android - 08/06/2021

Quem sou eu?

Haldny Santos

Engenheiro Android - XP Inc

Desenvolvimento Android +7 Anos

Desenvolvimento de Software +11 Anos

Membro da comunidade AndroidDevBR

Onde me encontrar?

Agenda

- O que é um Gradle Plugin?
- Onde criar o Gradle Plugin?
- Publicando o Plugin no Gradle Plugin Portal
- Arquitetura do aplicativo
- Casos de uso:
  - 1 - Versionamento das bibliotecas e das aplicações
    - Entendendo como eram feitos os pull-requests
    - Criando um plugin para automatizar o versionamento
  - 2 - Configurações iniciais dos módulos e aplicativos
    - Configurações iniciais e problemas enfrentados
    - Criando um plugin para padronizar as configurações
- Referências
- Agradecimentos e Perguntas

O que é um Gradle Plugin?

- São partes de código, geralmente reutilizáveis, de lógica de construção dos artefatos;
- Podem ser utilizados em muitos projetos e construções diferentes;
- O Gradle permite criarmos nossos plugins e compartilharmos com outras pessoas;
- Pode ser desenvolvido em qualquer linguagem que termine compilada como bytecode JVM;
- Geralmente são implementados nas linguagens Groovy, Kotlin, Java;
- Plugins desenvolvidos em Kotlin ou Java tem desempenho melhor que os desenvolvidos em Groovy por serem linguagens estaticamente tipadas;

Onde criar o Gradle Plugin?

Existem alguns lugares onde podemos criar o nosso plugin, eles são:

1. Build Script;
2. buildSrc do projeto;
3. Projeto standalone;

Build script

Você pode incluir o código-fonte do plugin diretamente no script de construção. Isso tem a vantagem de que o plug-in é compilado automaticamente e incluído no classpath do script de construção sem que você precise fazer nada. No entanto, o plugin não fica visível fora do script de construção e, portanto, você não pode reutilizá-lo fora do script de construção em que ele está definido.

Build script (exemplo)

build.gradle.kts

```
android {}
...
dependencies {}
...
class GreetingPlugin : Plugin<Project> {
   override fun apply(project: Project) {
       project.task("hello") {
           doLast {
               println("Hello from the GreetingPlugin")
           }
       }
   }
}
apply<GreetingPlugin>()
```

Build script (exemplo)

MAC01/PROJETO1/U1$ **./gradlew hello**

Configuration on demand is an incubating feature.

> Task :projeto1:hello

Hello from the GreetingPlugin

buildSrc do projeto

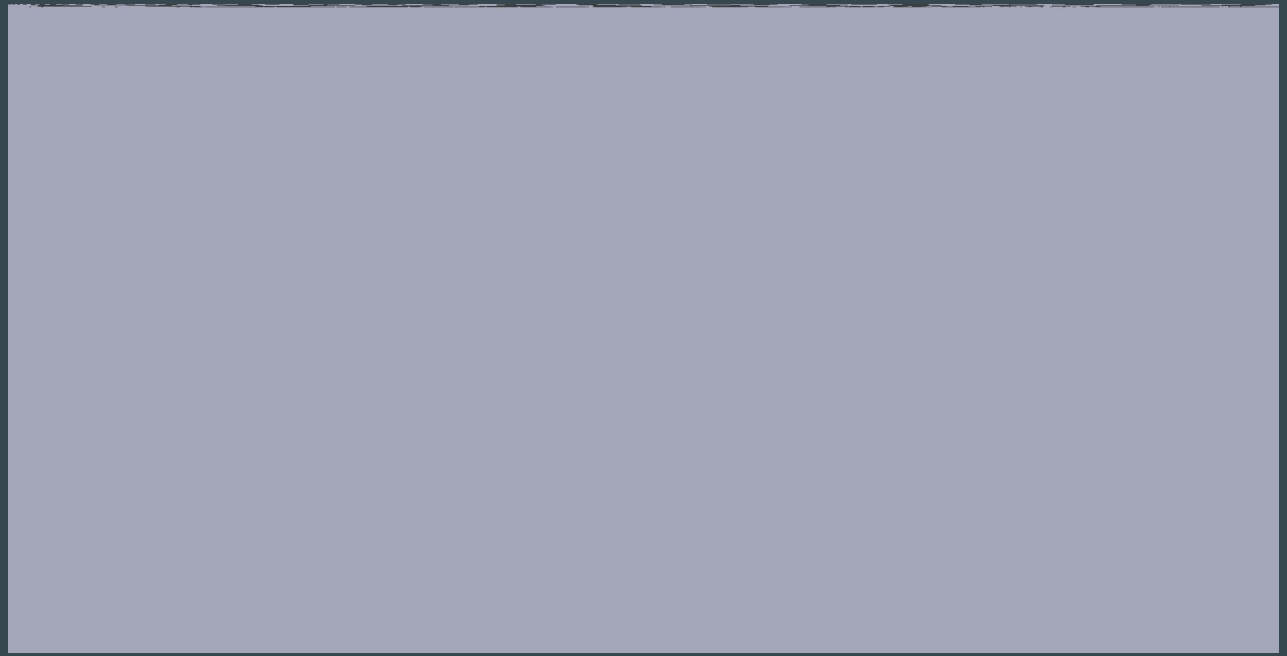
Dependendo da linguagem escolhida, você pode incluir o código-fonte do plugin nos seguintes diretórios:

- **rootProjectDir/buildSrc/src/main/java (Linguagem Java);**
- **rootProjectDir/buildSrc/src/main/kotlin (Linguagem Kotlin);**
- **rootProjectDir/buildSrc/src/main/groovy (Linguagem Groovy).**

O Gradle se encarregará de compilar, testar e tornar o plugin disponível no classpath do script de construção, ou seja, o plugin é visível para todos os scripts de construção (módulo principal e todos os submódulos). No entanto, ele não é visível fora da construção e, portanto, você não pode reutilizar o plugin fora da construção em que ele está definido.

buildSrc do projeto (exemplo)

- Criar o diretório buildSrc



buildSrc do projeto (exemplo)

- Criar o arquivo build.gradle.kts dentro do diretório buildSrc

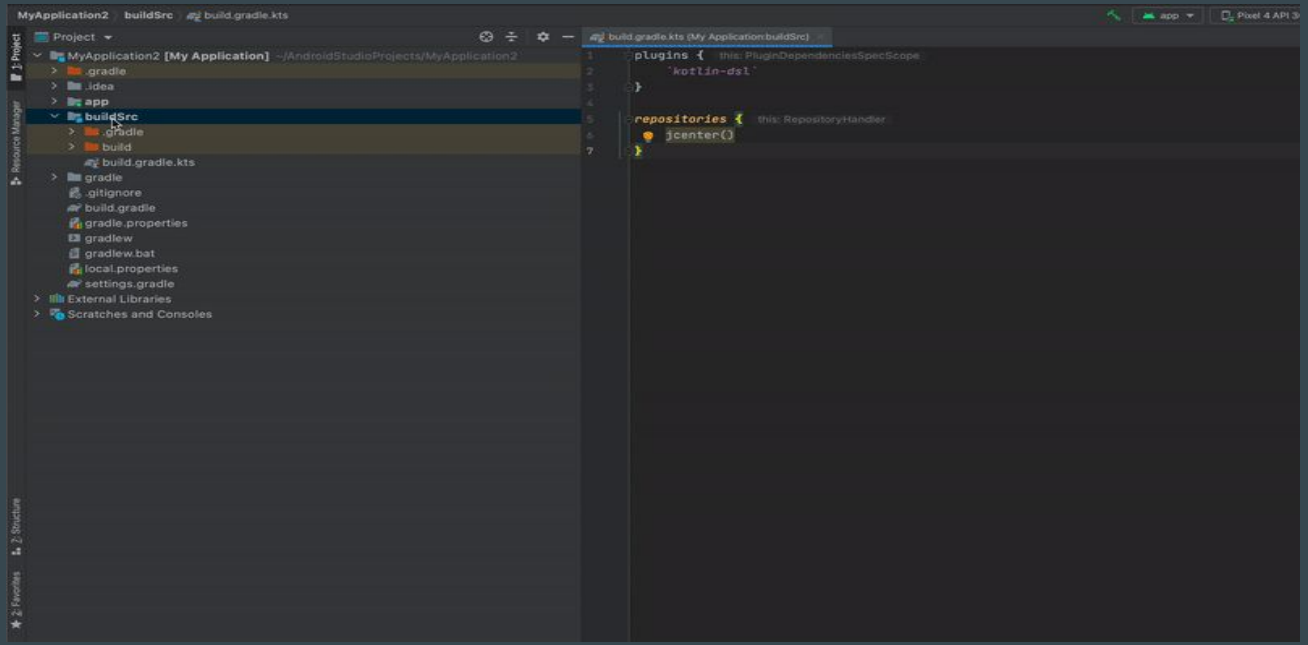
build.gradle.kts

```
plugins {
    `kotlin-dsl`
}


repositories {
    jcenter()
}
```

- Após isso, clicar no "Sync now"

buildSrc do projeto (exemplo)

- Criar o diretório do código-fonte



buildSrc do projeto (exemplo)

- Criar a classe no pacote desejado

```
package com.example.haldny

import org.gradle.api.Plugin
import org.gradle.api.Project

class GreetingPlugin : Plugin<Project> {
   override fun apply(project: Project) {
       project.task("hello") {
           doLast {  println("Hello from the GreetingPlugin") }
       }
   }
}
```

buildSrc do projeto (exemplo)

- Alterar o arquivo build.gradle.kts para criar o plugin no maven local

```
plugins {
    `kotlin-dsl`
    `maven`
}

repositories {
    jcenter()
}
...
```

buildSrc do projeto (exemplo)

- Alterar o arquivo build.gradle.kts para criar o plugin no maven local

```
...
group "com.example.haldny"
version "0.0.1"

gradlePlugin {
    plugins {
        create("hello") {
            id = "hello"
            implementationClass = "com.example.haldny.GreetingPlugin"
        }
    }
}
```

buildSrc do projeto (exemplo)

- Aplicar o plugin no build.gradle.kts da aplicação

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'hello'
}
```

buildSrc do projeto (exemplo)

MAC01/APP1/U1$ **./gradlew hello**

> Task :app:hello

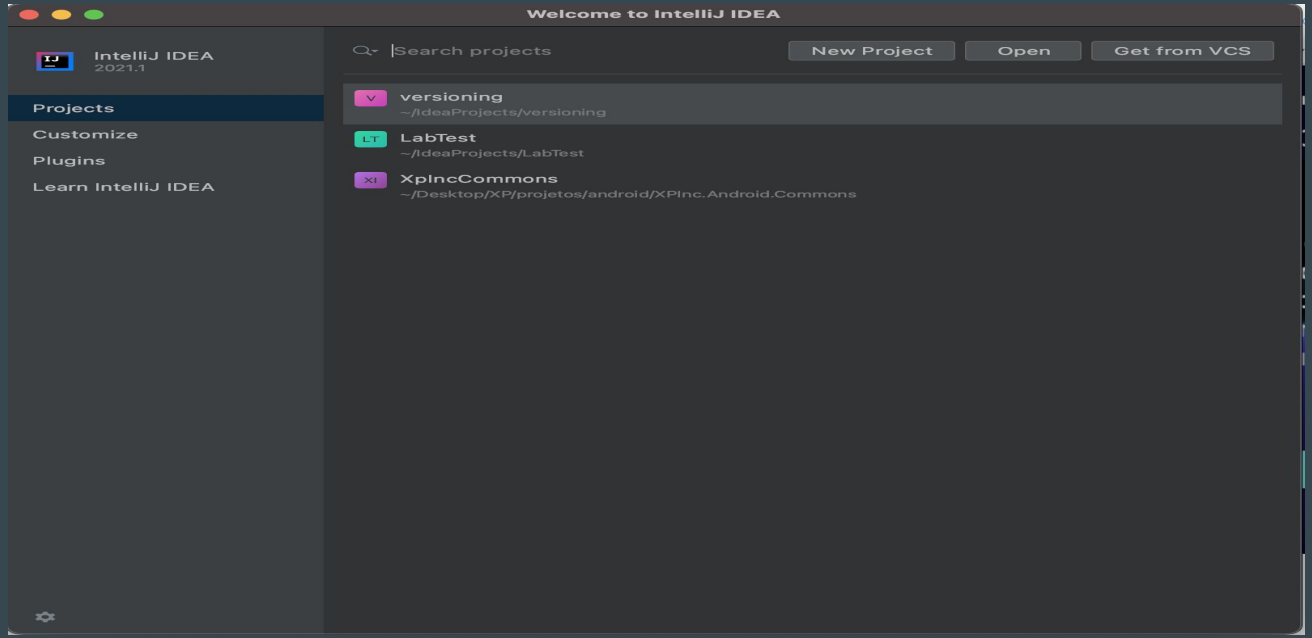
Hello from the GreetingPlugin

Projeto Standalone

Você pode criar um projeto separado para o seu plugin. Este projeto produz e publica um **JAR** que você pode usar em várias compilações e compartilhar com outras pessoas. Geralmente, este **JAR** pode incluir alguns plugins ou agrupar várias classes de tarefas relacionadas em uma única biblioteca, ou, até mesmo ambos.

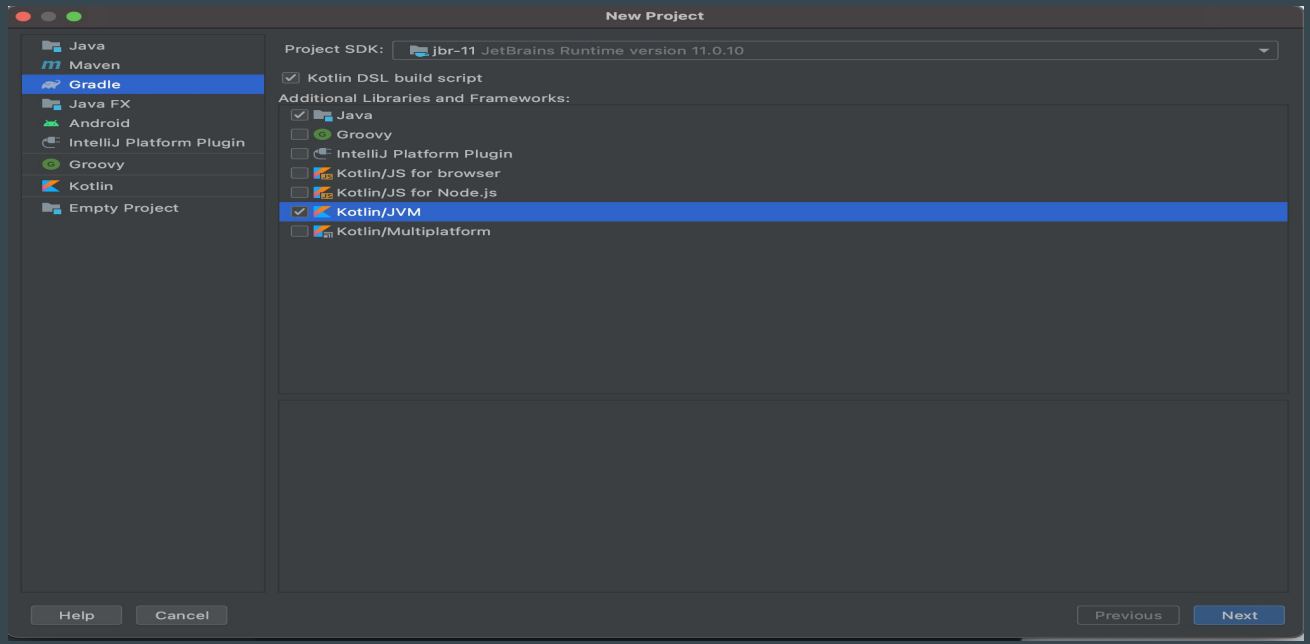
Projeto Standalone (exemplo)

- Criando o projeto no IntelliJ



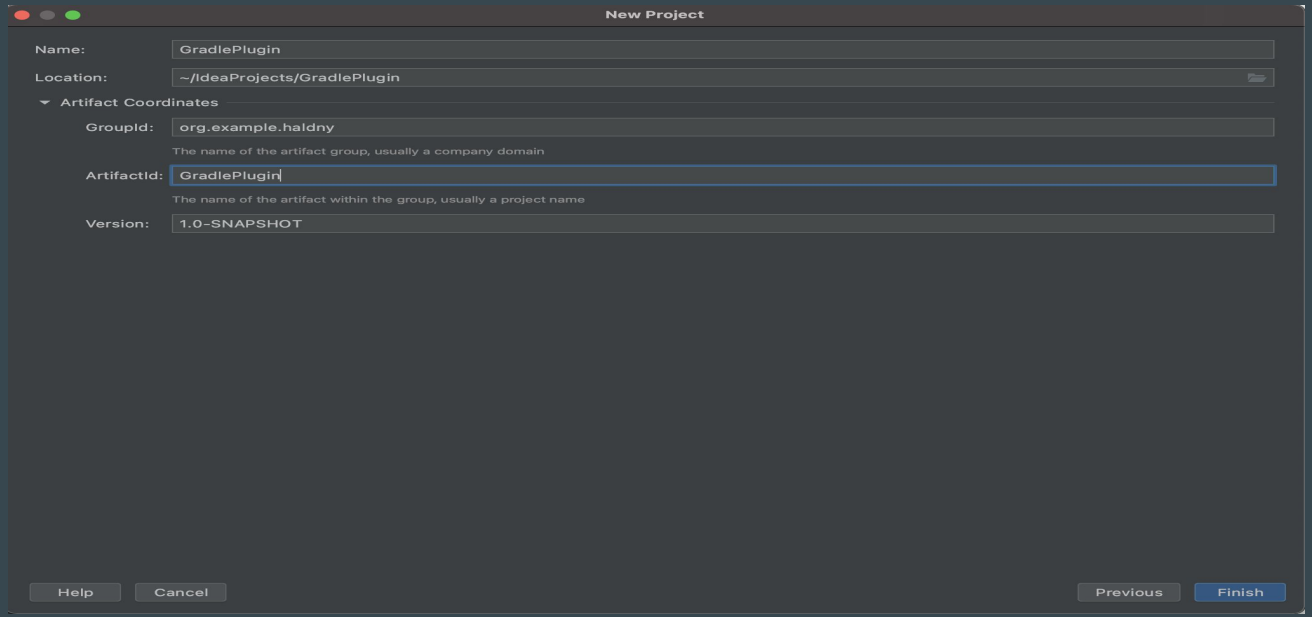
Projeto Standalone (exemplo)

- Criando o projeto no IntelliJ



Projeto Standalone (exemplo)

- Criando o projeto no IntelliJ



Projeto Standalone (exemplo)

build.gradle.kts

```
plugins {
    java
    kotlin("jvm") version "1.4.32"
    `java-gradle-plugin`
    `maven`
}

group = "org.example.haldny"
version = "1.0-SNAPSHOT"
...
```

Projeto Standalone (exemplo)

build.gradle.kts

```
repositories {
    mavenCentral()
}

dependencies {
    implementation(kotlin("stdlib"))
}
...
```

Projeto Standalone (exemplo)

build.gradle.kts

```
gradlePlugin {
    plugins {
        create("hello2") {
            id = "org.example.haldny.hello2"
            implementationClass = "org.example.haldny.GradlePlugin"
        }
    }
}
```

Projeto Standalone (exemplo)

- Criar a classe no pacote desejado

```
package org.example.haldny

import org.gradle.api.Plugin
import org.gradle.api.Project

class GradlePlugin : Plugin<Project> {
   override fun apply(project: Project) {
       project.task("hello2") {
           it.doLast {  println("Hello from the GradlePlugin") }
       }
   }
}
```

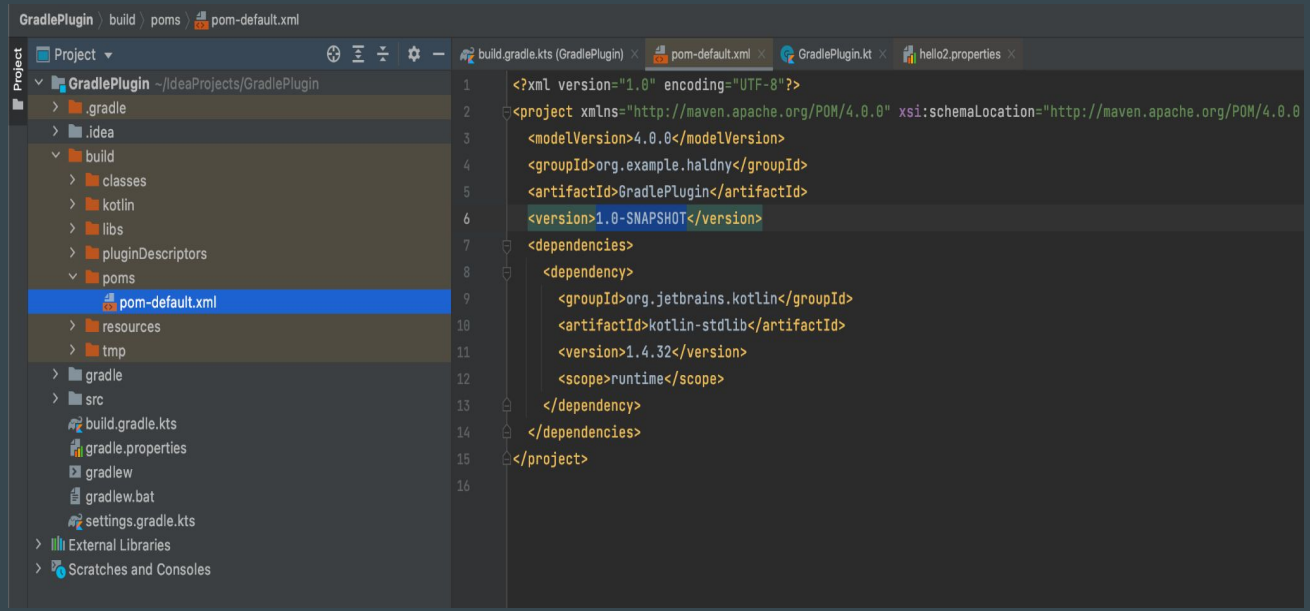
Projeto Standalone (exemplo)

- Instalar o plugin no nosso mavel local

```
MAC01:GradlePlugin u1$ ./gradlew install


BUILD SUCCESSFUL in 938ms
```

Projeto Standalone (exemplo)

- Arquivo pom.xml



Projeto Standalone (exemplo)

- Aplicar o plugin no build.gradle.kts da aplicação

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'hello'
    id 'org.example.haldny.hello2'
}
```

Projeto Standalone (exemplo)

- Configurar o plugin no build.gradle.kts do projeto

```
buildscript {
    repositories {
        ...
        mavenCentral()
    }
    dependencies {
        ...
        classpath "org.example.haldny:GradlePlugin:1.0-SNAPSHOT"
    }
}
```

Projeto Standalone (exemplo)

MAC01/APP1/U1$ **./gradlew hello2**

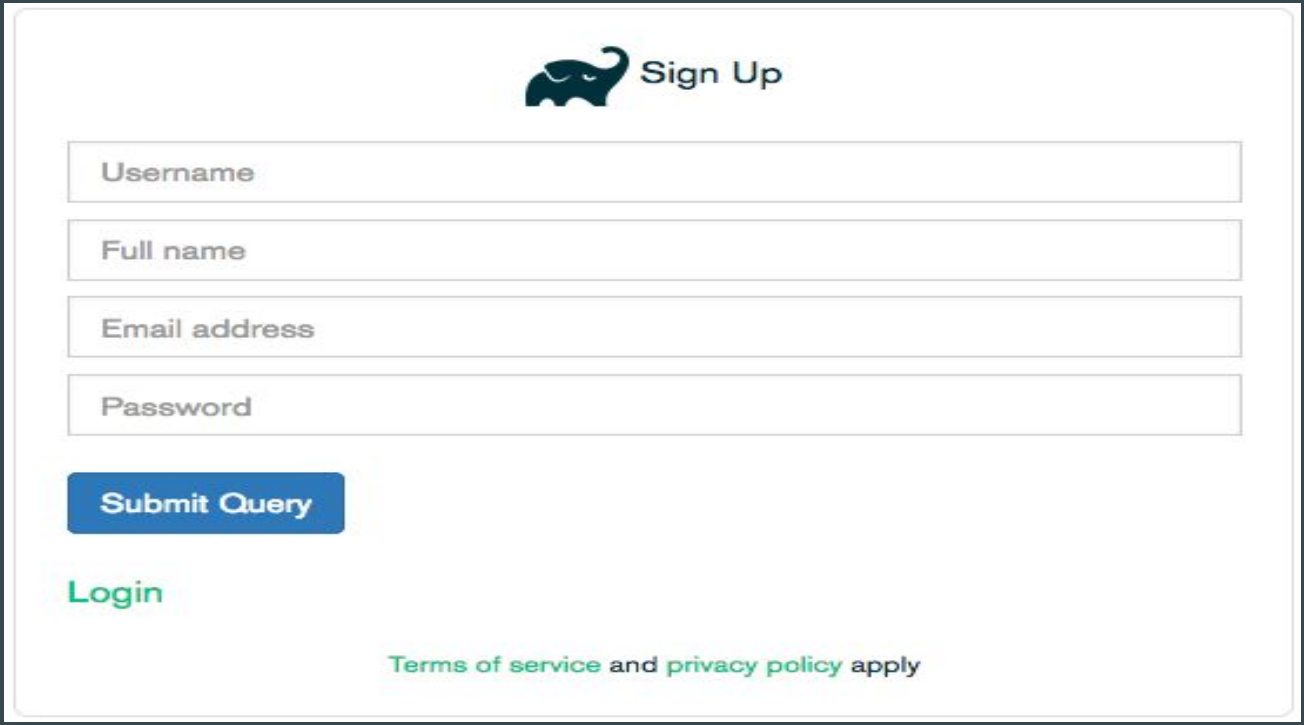
> Task :app:hello2

Hello from the GradlePlugin

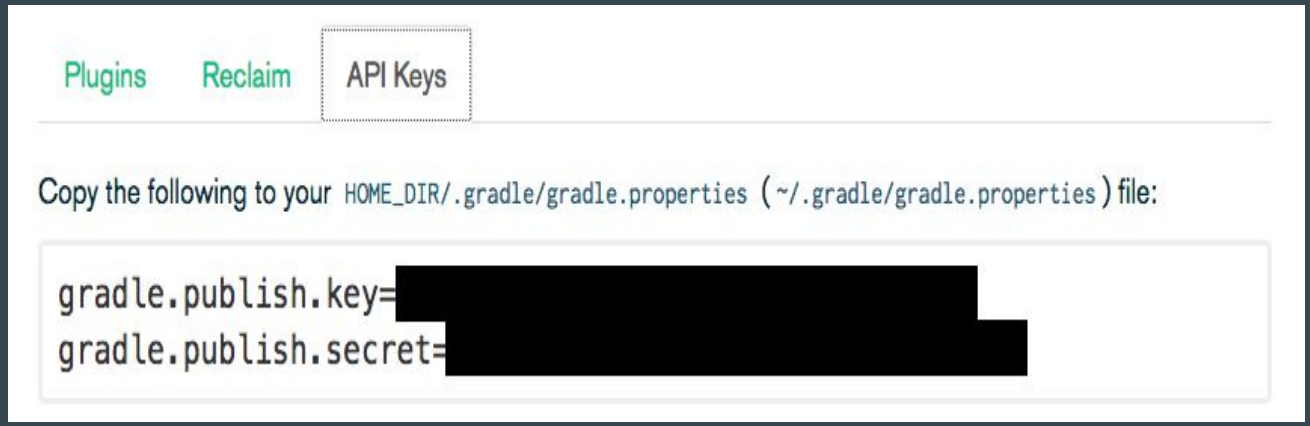
Publicando o Plugin no Gradle Plugin Portal

1. Criar uma conta do Gradle Plugin Portal;
2. Pegar a API KEY e SECRET;
3. Configurar o seu projeto standalone;

Criar uma conta no Gradle Plugin Portal



Pegar API KEY e SECRET



Configurar seu projeto standalone

```
plugins {
    java
    kotlin("jvm") version "1.4.32"
    `java-gradle-plugin`
    `maven`
    `maven-publish`
     id("com.gradle.plugin-publish") version "0.14.0"
}
```

Configurar seu projeto standalone

```
pluginBundle {

    website = "https://github.com/haldny/GradlePlugin"

    vcsUrl = "https://github.com/haldny/GradlePlugin"

    tags = listOf("master", "develop")

}
```

Configurar seu projeto standalone

```
gradlePlugin {
    plugins {
        create("hello2") {
            id = "org.example.haldny.hello2"
            implementationClass = "org.example.haldny.GradlePlugin"
            displayName = "Gradle Plugin"
            description = "Template for people to start their own plugin adventure"
        }
    }
}
```

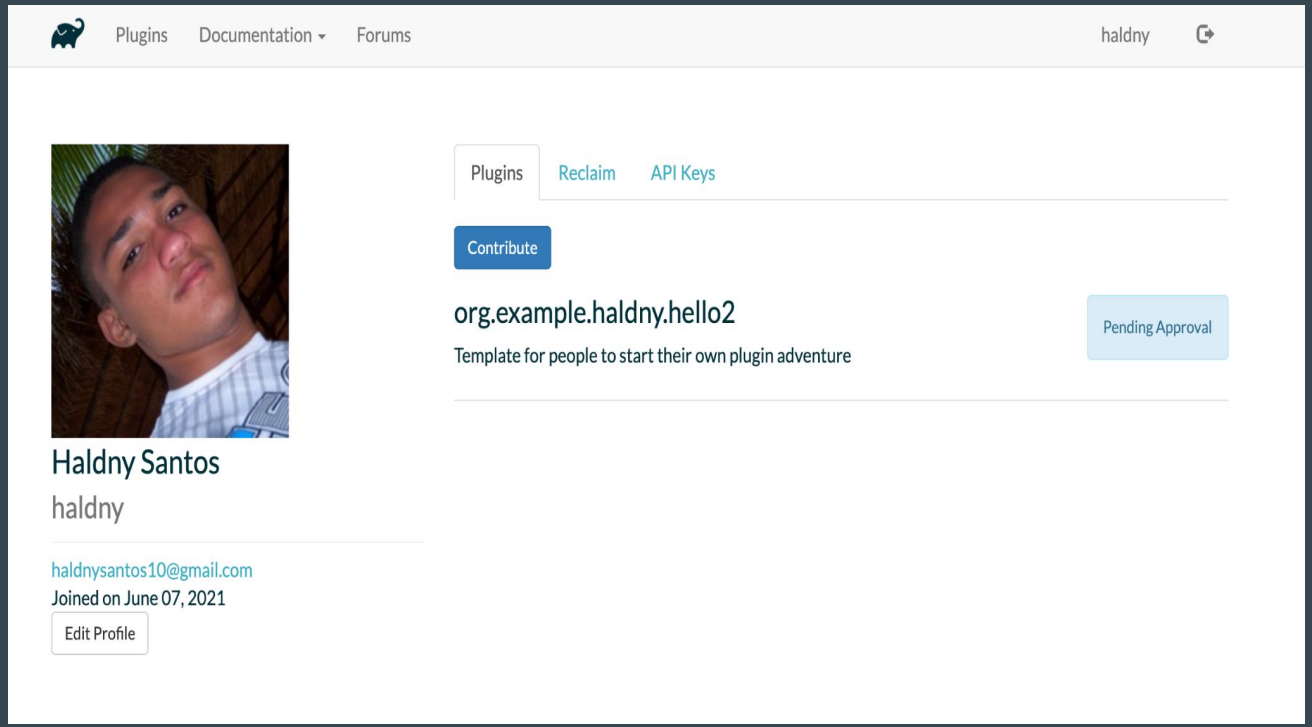
Publicando seu projeto standalone

```
MAC01:GradlePlugin u1$ ./gradlew publishPlugins

> Task :publishPlugins
Publishing plugin org.example.haldny.hello2 version 1.0-SNAPSHOT
Thank you. Your new plugin org.example.haldny.hello2 has been submitted for approval by Gradle engineers. The request should be processed within the next few days, at which point you will be contacted via email.
Publishing artifact build/libs/GradlePlugin-1.0-SNAPSHOT.jar
Publishing artifact build/libs/GradlePlugin-1.0-SNAPSHOT.jar
Publishing artifact build/publications/pluginMaven/pom-default.xml
Publishing artifact build/publications/pluginMaven/module.json
Activating plugin org.example.haldny.hello2 version 1.0-SNAPSHOT

BUILD SUCCESSFUL in 5s
```

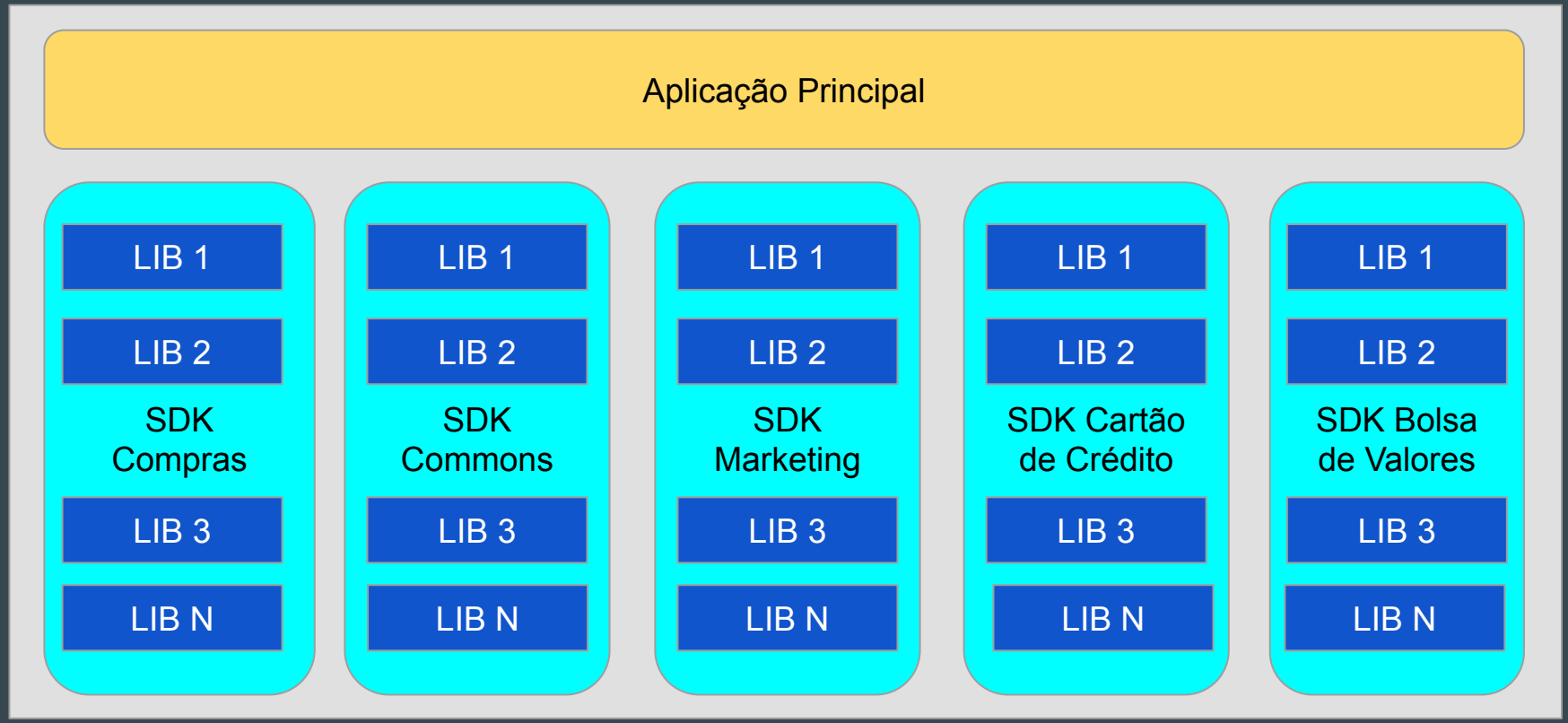
Ou

```
$ ./gradlew publishPlugins -Pgradle.publish.key=<key> -Pgradle.publish.secret=<secret>
```

Publicando seu projeto standalone



Arquitetura do Aplicativo



Casos de uso

1. Versionamento das bibliotecas e das aplicações

Problema encontrado:

- O versionamento era feito de forma manual, tendo o programador a atribuição de subir a versão quando seu código tiver sido aprovado;
- Era necessário fazer uma atualização no commit alterando a versão após a aprovação, ou seja, era necessário a aprovação desse novo patch gerando retrabalho.

Casos de uso

1. Versionamento das bibliotecas e das aplicações

Solução:

- Criar um plugin para fazer a atualização das versões de forma automática com base no número de merges feitos;

Plugin de Versionamento

```
class VersioningPlugin : Plugin<Project> {

    override fun apply(project: Project) {

        with(project) {

            extensions.create("versioning", VersioningPluginExtension::class.java)

        }

    }

}
```

Plugin de Versionamento

```
open class VersioningPluginExtension {
    var versionFilePath: String? = null
    var useCustomVersioning: Boolean = false
    fun getVersionCode(): Int? {
        val versionProperties = getVersionProperties()
        val versioning = Versioning.getVersioningInstance(versionProperties, useCustomVersioning)
        return versioning.getVersionCode()
    }
    fun getVersionName(): String? {
        val versionProperties = getVersionProperties()
        val versioning = Versioning.getVersioningInstance(versionProperties, useCustomVersioning)
        return versioning.getVersionName()
    }
}
```

Plugin de Versionamento

```
abstract class Versioning {
    abstract fun getVersionCode() : Int?
    abstract fun getVersionName() : String?
    companion object {
        fun getVersioningInstance(
            versionProperties: Properties?,
            useCustomVersioning: Boolean
        ): Versioning {
            return when (getPropertyType(versionProperties, useCustomVersioning)) {
                PropertyType.DEFAULT -> DefaultVersioning()
                PropertyType.USE_SAME_STRATEGY -> SameStrategyVersioning(versionProperties)
                PropertyType.USE_CUSTOM_STRATEGY -> CustomStrategyVersioning(versionProperties)
            }
        }
    }
```

Plugin de Versionamento

```
class DefaultVersioning() : Versioning() {

    override fun getVersionCode() = getCommitCount().apply { println("VersionCode $this") }

    override fun getVersionName(): String {
        return getCommitCount().toString().apply { println("VersionName $this") }
    }

}
```

Plugin de Versionamento

```
class SameStrategyVersioning(private val versionProperties: Properties?) : Versioning() {

   override fun getVersionCode(): Int? {

       val majorVersion = versionProperties?.getProperty("VERSION_MAJOR", "0").toString().toInt()

       val minorVersion = versionProperties?.getProperty("VERSION_MINOR", "0").toString().toInt()

       val buildBase = versionProperties?.getProperty("VERSION_BUILD_BASE","0").toString().toInt()

       val commitCount = getCommitCount()

       val buildNo = commitCount - buildBase

       return Integer.parseInt(String.format("%02d%02d%03d", majorVersion, minorVersion, buildNo))

   }

...
```

Plugin de Versionamento

```
    override fun getVersionName(): String? {

        val majorVersion = versionProperties?.getProperty("VERSION_MAJOR", "0").toString().toInt()

        val minorVersion = versionProperties?.getProperty("VERSION_MINOR", "0").toString().toInt()

        val buildBase = versionProperties?.getProperty("VERSION_BUILD_BASE","0").toString().toInt()

        val commitCount = getCommitCount()

        val buildNo = commitCount - buildBase


        return String.format("%02d.%02d.%03d", majorVersion, minorVersion, buildNo)

    }

}
```

Plugin de Versionamento

```
class CustomStrategyVersioning(private val versionProperties: Properties?) : Versioning() {


    override fun getVersionCode(): Int? {

        val baseVersionCode =  versionProperties?.getProperty("VERSION_BASE_CODE",
"0").toString().toInt()

        val baseCommitHash =
versionProperties?.getProperty("VERSION_BASE_COMMIT_HASH").toString()


        val commitCount = internalCommitCount(baseCommitHash)

        return baseVersionCode + commitCount

    }
```

Plugin de Versionamento

```
    override fun getVersionName(): String? {

        val majorVersion = versionProperties?.getProperty("VERSION_MAJOR", "0").toString().toInt()

        val minorVersion = versionProperties?.getProperty("VERSION_MINOR", "0").toString().toInt()

        val buildBase = versionProperties?.getProperty("VERSION_BUILD_BASE", "0").toString().toInt()

        val commitCount = getCommitCount()

        val buildNo = commitCount - buildBase


        return String.format("%02d.%02d.%03d", majorVersion, minorVersion, buildNo)

    }

}
```

Plugin de Versionamento

- Instalar o plugin no nosso mavel local

```
MAC01:versioning u1$ ./gradlew install


BUILD SUCCESSFUL in 958ms
```

Plugin de Versionamento

- Aplicar o plugin no build.gradle.kts da aplicação

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'hello'
    id 'com.example.haldny.versioning'
}
android {
     versionCode versioning.getVersionCode()
     versionName versioning.getVersionName()
}
```

Projeto Standalone (exemplo)

- Configurar o plugin no build.gradle.kts do projeto

```
buildscript {
    repositories {
        ...
        mavenCentral()
    }
    dependencies {
        ...
        classpath "org.example.haldny:versioning:1.0-SNAPSHOT"
    }
}
```

Casos de uso

2. Configurações iniciais dos módulos e aplicativos

Problema encontrado:

- Cada módulo ou aplicativo, tinha toda a sua configuração definida de forma repetida ou copiada de outro módulo;
- Tínhamos diversas SDKs sendo compiladas com diferentes versões do Android;
- Muito código repetido;

Casos de uso

2. Configurações iniciais dos módulos e aplicativos

Solução:

- Criar um plugin para fazer as configurações iniciais de módulos e aplicativos, de forma a padronizar e evitar configurações repetidas e/ou erradas;

Plugin de Configuração

```
class ConfiguringProjectPlugin : Plugin<Project> {

    override fun apply(project: Project) {

        applyDefaultPlugins(project)

        val androidExtension = project.extensions.getByName("android")

        if (androidExtension is BaseExtension) {

            applyDefaultAndroidConfig(androidExtension)

            project.tasks.withType(KotlinCompile::class.java).configureEach {

                it.kotlinOptions { jvmTarget = "1.8" }

            }

            configureProject(project, androidExtension)

        }

    }
```

Plugin de Configuração

```
private fun configureProject(project: Project, androidExtension: BaseExtension) {

    val proguardFile = "proguard-rules.pro"

    androidExtension.apply {

        when (this) {

            is LibraryExtension -> configureAndroidLibrary(project, androidExtension as
LibraryExtension, proguardFile)

            is AppExtension -> configureAppAndroid(project, androidExtension as AppExtension,
proguardFile)

        }

    }

}
```

Plugin de Configuração

```
private fun applyDefaultAndroidConfig(androidExtension: BaseExtension) {
    androidExtension.compileOptions {
        it.sourceCompatibility = JavaVersion.VERSION_1_8
        it.targetCompatibility = JavaVersion.VERSION_1_8
    }
    androidExtension.apply {
        compileSdkVersion(30)
        defaultConfig {
            it.targetSdkVersion(30)
            it.minSdkVersion(23)
            it.versionCode = 100
            it.versionName = "1.0.0"
            it.testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        }
    }
}
```

Plugin de Configuração

```
private fun configureAppAndroid(project: Project, appExtension: AppExtension, proguardFile: String) {
    project.plugins.apply("com.android.application")
    appExtension.apply {
        signingConfigs {
            it.create("release") { signingConfigs ->
                signingConfigs.isV2SigningEnabled = true
            }
        }
        defaultConfig {
            it.applicationId = "applicationId.haldny"
        }
...
```

Plugin de Configuração

```
buildTypes {
            it.getByName("release") { buildType ->
                buildType.isMinifyEnabled = true
                buildType.isShrinkResources = true
                buildType.proguardFiles(
                    getDefaultProguardFile("proguard-android-optimize.txt"),
                    proguardFile
                )
            }
        }
    }
}
```

Plugin de Configuração

```
private fun configureAndroidLibrary(project: Project, libraryExtension: LibraryExtension, proguardFile:
String) {

    project.plugins.apply("com.android.library")

    libraryExtension.apply {

        buildTypes {

            it.getByName("release") { buildType ->

                buildType.isDebuggable = false

                buildType.isMinifyEnabled = false

                buildType.multiDexEnabled = false

                buildType.consumerProguardFile("dexguard-project.txt")

                buildType.consumerProguardFile("proguard-rules.pro")

                buildType.consumerProguardFile("consumer-rules.pro")

            }

        }

    }

}
```

Plugin de Versionamento

- Instalar o plugin no nosso mavel local

```
MAC01:configuring u1$ ./gradlew install


BUILD SUCCESSFUL in 1024ms
```

Plugin de Versionamento

- Aplicar o plugin no build.gradle.kts da aplicação

```
plugins {
   id 'com.example.haldny.configuring'
}
android {
    ...
}
```

Projeto Standalone (exemplo)

- Configurar o plugin no build.gradle.kts do projeto

```
buildscript {
    repositories {
        ...
        mavenCentral()
    }
    dependencies {
        ...
        classpath "org.example.haldny:configuring:1.0-SNAPSHOT"
    }
}
```

Referências

- https://docs.gradle.org/current/userguide/publishing_gradle_plugins.html
- https://docs.gradle.org/current/userguide/custom_plugins.html
- https://dzone.com/articles/the-complete-custom-gradle-plugin-building-tutoria
- https://docs.gradle.org/current/userguide/publishing_gradle_plugins.html
- https://medium.com/wantedly-engineering/managing-android-multi-module-project-with-gradle-plugin-and-kotlin-4fcc126e7e49
- https://github.com/malvinstn/gradle-plugin-android-multi-module

Agradecimentos

Perguntas?